

# R600 ISA

Dresden, 30.11.2011

# Goals

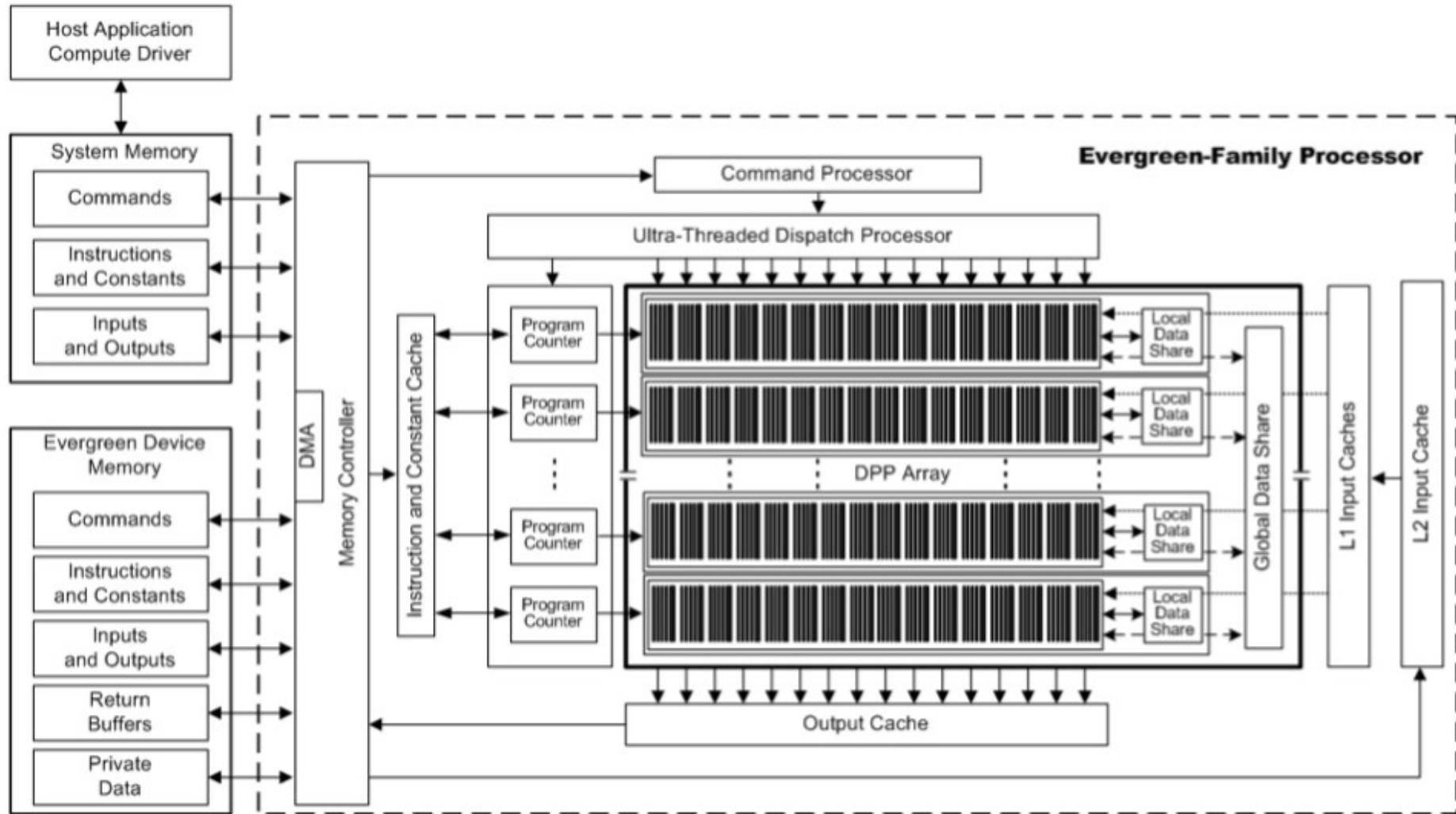
- Give an overview to GPU ISA
- Knowing which programs run faster than others
- Preparation to read the official documentation from AMD

## History

- R600 is the chip used in Radeon HD 2000/3000 cards and FireGL 2007 series
- Introduced unified shader architecture for PC
- Consider R600 as a massive multicore CPU where each core has massive hyper threading

Source: [http://en.wikipedia.org/wiki/Radeon\\_R600](http://en.wikipedia.org/wiki/Radeon_R600)

# Block diagram of the R600 processor



Source: [http://developer.amd.com/sdks/AMDAAPPSDK/assets/AMD\\_Evergreen-Family\\_Instruction\\_Set\\_Architecture.pdf](http://developer.amd.com/sdks/AMDAAPPSDK/assets/AMD_Evergreen-Family_Instruction_Set_Architecture.pdf)

# Concepts

- Command Processor (CP)
  - Ring buffer
  - Indirect buffer 0
  - Indirect buffer 1
- Pipelines
  - Vertices
  - Geometry
  - Fragments
- Wavefronts
  - 64 threads run with the same program counter
  - Control flow contains loop instructions
  - Each thread has its own data and ALU
  - IF/ELSE with execution mask
- Memory access from shaders
  - Vertex fetch (Buffers)
  - Texture fetch (Textures)
  - RAT (available since r800)

# Program Types

- Vertex Shader
- Geometry Shader
- DMA Copy
- Pixel Shader
- New to r800:
  - Compute Shaders
  - Hull Shader
  - Domain Shader

# Thread Organization

- 'The R600 processor hides memory latency by keeping track of potentially hundreds of threads in different stages of execution, and by overlapping compute operations with memory-access operations.' (source: r600isa.pdf)
- Thread state consists of
  - GPRs
  - CRs
  - Temp registers for ALU, VTX and TX clauses
  - Execution mask

# Control Flow Programs

- One instruction: 64 bits
- Call ALU clauses (ALU), texture fetch clauses and vertex fetch clauses (VTX)
- import and export data
- Functions to emit vertices, primitives and such
- Write and read on ring buffers, scratch buffers, reduction buffers, stream buffers
- Loops with LOOP\_BEGIN, LOOP\_BREAK, LOOP\_CONTINUE and LOOP\_END and a loop count (can be nested)
- PUSH, POP, ELSE, JUMP
  - Manipulate execution mask
  - Execution mask can predicate instruction execution
  - JUMPs speed up the program: they can skip instructions when all threads are have a certain flag in the execution mask
- Subprograms with CALL and RETURN
- END\_OF\_PROGRAM

**Table 2.7 Flow of a Typical Program**

Function	Microcode Formats <sup>1</sup>	
	Control Flow (CF) Code	Clause Code
Start loop.	CF_DWORD[0,1]	
Initiate a fetch through a texture cache clause.	CF_DWORD[0,1]	
Fetch through a texture cache or vertex cache clause to load data from memory to GPRs.		TEX_DWORD[0,1,2]
Initiate ALU clause.	CF_ALU_DWORD[0,1]	
ALU clause to compute on loaded data and literal constants. This example shows a single clause consisting of a single ALU <i>instruction group</i> containing five ALU instructions (two quadwords each) and two quadwords of literal constants.		ALU_DWORD[0,1] ALU_DWORD[0,1] ALU_DWORD[0,1] ALU_DWORD[0,1] ALU_DWORD[0,1] LAST bit set Literal[X,Y] Literal[Z,W]
End loop.	CF_DWORD[0,1]	
Allocate space in an output buffer.	CF_ALLOC_EXPORT_DWORD0 CF_ALLOC_EXPORT_DWORD1_BU F	
Export (write) results from GPRs to output buffer.	CF_ALLOC_EXPORT_DWORD0 CF_ALLOC_EXPORT_DWORD1_BU F	

Source: [http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD\\_Evergreen-Family\\_Instruction\\_Set\\_Architecture.pdf](http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_Evergreen-Family_Instruction_Set_Architecture.pdf)

# ALU Clauses

- Up to 5 slots (X, Y, Z, W, Trans)
- Transcendent slot can perform more complex operations
- 0, 2 or 4 literals
- 64 bits per instruction
- Can access 128 GPRs and 256 constants
- Call from CF with ALU or PRED ALU

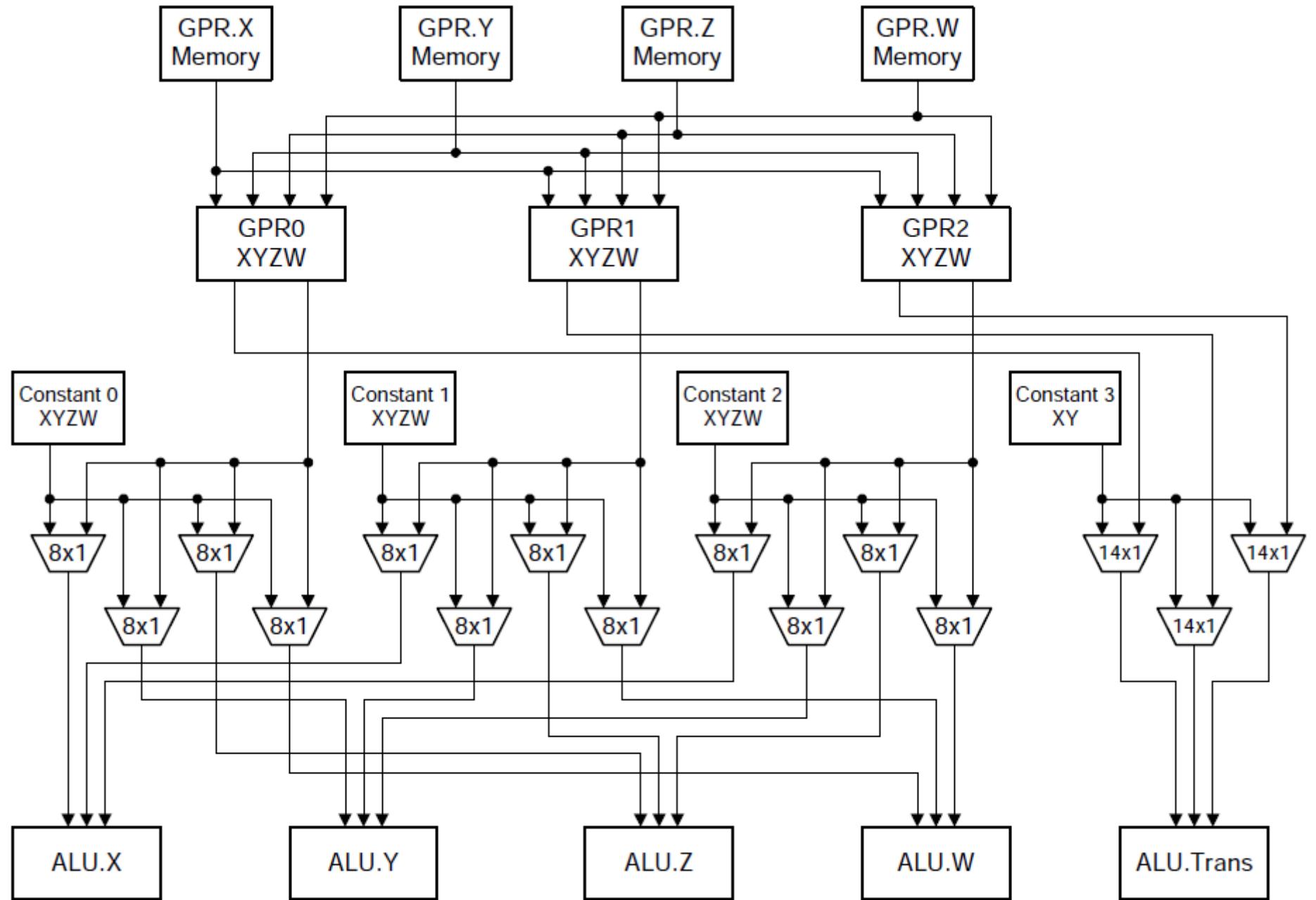
```
0000 00000000 CF ADDR:0
0001 84C00000 CF INST:19 COND:0 POP_COUNT:0
0002 00000004 ALU ADDR:8 KCACHE_MODE0:0 KCACHE_BANK0:0 KCACHE_BANK1:0
0003 A01C0000 ALU INST:64 KCACHE_MODE1:0 KCACHE_ADDR0:0 KCACHE_ADDR1:0 COUNT:8
0008 00000001 SRC0(SEL:1 REL:0 CHAN:0 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0009 00600C90 INST:25 DST(SEL:3 CHAN:0 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0010 00000401 SRC0(SEL:1 REL:0 CHAN:1 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0011 20600C90 INST:25 DST(SEL:3 CHAN:1 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0012 00000801 SRC0(SEL:1 REL:0 CHAN:2 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0013 40600C90 INST:25 DST(SEL:3 CHAN:2 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0014 80000C01 SRC0(SEL:1 REL:0 CHAN:3 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:1)
0015 60600C90 * INST:25 DST(SEL:3 CHAN:3 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0016 00000002 SRC0(SEL:2 REL:0 CHAN:0 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0017 00800C90 INST:25 DST(SEL:4 CHAN:0 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0018 00000402 SRC0(SEL:2 REL:0 CHAN:1 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0019 20800C90 INST:25 DST(SEL:4 CHAN:1 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0020 00000802 SRC0(SEL:2 REL:0 CHAN:2 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:0)
0021 40800C90 INST:25 DST(SEL:4 CHAN:2 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0022 80000C02 SRC0(SEL:2 REL:0 CHAN:3 NEG:0) SRC1(SEL:0 REL:0 CHAN:0 NEG:0) LAST:1)
0023 60800C90 * INST:25 DST(SEL:4 CHAN:3 REL:0 CLAMP:0) BANK_SWIZZLE:0 SRC0_ABS:0 SRC1_ABS:0 WRITE_MASK:1 OMOD:0 EXECUTE_MASK:0 UPDATE_PRED:0
0004 C001A03C EXPORT GPR:3 ELEM_SIZE:3 ARRAY_BASE:3C TYPE:1
0005 95000688 EXPORT SWIZ_X:0 SWIZ_Y:1 SWIZ_Z:2 SWIZ_W:3 BARRIER:1 INST:84 BURST_COUNT:1 EOP:0
0006 C0024000 EXPORT GPR:4 ELEM_SIZE:3 ARRAY_BASE:0 TYPE:2
0007 95200688 EXPORT SWIZ_X:0 SWIZ_Y:1 SWIZ_Z:2 SWIZ_W:3 BARRIER:1 INST:84 BURST_COUNT:1 EOP:1
```

# Data layout of ALU instructions

- Source selection flags
  - Read from GPR
  - Read from constant bank
  - Read a previous result
  - Load a literal constant
  - Load float 0.0, 0.5 or 1.0
  - Load integer -1, 0, 1
  - Set ABS and/or NEG bit
- Destination GPR
- Instruction
- Output modifier
- CLAMP bits

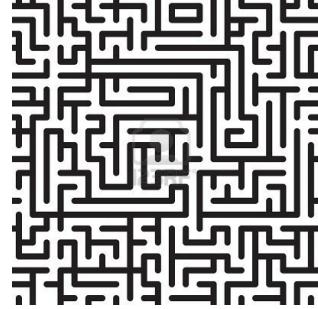
C	D E	D R	DST_GPR			B S	ALU_INST			O MOD	F M	W M	U P	U E M	S 1 A	S 0 A	+4
L	P S	I M	S 1 N	S 1 E	S 1 R	SRC1_SEL			S 0 N	S 0 E	S 0 R	SRC0_SEL				+0	

Source: <http://x.org/docs/AMD/r600isa.pdf>



**Figure 4-3. ALU Data Flow**

# ALU Instructions



- ADD\_INT, AND\_INT, MUL, MUL\_IEEE, MULADD, MULADD\_D2, MULADD\_D4, MULADD\_IEEE\_D2
- MOV, CMOV??\_INT, PRED\_SET??\_INT, SET??\_INT, CMOV??, PRED\_SET??, SET??
- MIN, MAX, TRUNC, CEIL
- Restricted to XYZW (no Trans)
  - DOT4, DOT4\_IEEE, MAX4
- Restricted to Trans unit
  - ASHR\_INT, INT\_TO\_FLT, MULLO\_INT, MULHI\_INT, RECIP\_UINT
  - SIN, COS, EXP\_IEEE, LOG\_CLAMPED, LOG\_IEEE
  - RECIP\_IEEE, RECIP\_FF, RECIP\_CLAMPED
  - MUL\_LIT\_D2, MUL\_LIT\_D4
  - RECIPSQRT\_CLAMPED, RECIPSQRT\_FF, RECIPSQRT\_IEEE

# Texture Fetch Clauses

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+12
S S W	S S Z	S S Y	S S X	SAMPLER_ID	OFFSET_Z	OFFSET_Y	OFFSET_X																					+8	
C T W	C T Z	C T Y	C T X	LOD_BIAS	D S W	D S Z	D S Y	D S X	D R	DST_GPR																+4			
Reserved				S R	SRC_GPR			RESOURCE_ID				F W Q	B F M	TEX_INST													+0		

Source: <http://x.org/docs/AMD/r600isa.pdf>

```

0000 00000002 TEX/VTX ADDR:4
0001 80800400 TEX/VTX INST:2 COUNT:2
0004 7C000000 INST:0 FETCH_TYPE:0 BUFFER_ID:0
0005 AC151001 SRC(GPR:0 SEL_X:0) MEGA_FETCH_COUNT:31 DST(GPR:1 SEL_X:0 SEL_Y:1 SEL_Z:2 SEL_W:5) USE_CONST_FIELDS:0 FORMAT(DATA:48 NUM:2 COMP:0 MODE:1)
0006 00080000 ENDIAN:0 OFFSET:0
0007 00000000
0008 7C000000 INST:0 FETCH_TYPE:0 BUFFER_ID:0
0009 AC151002 SRC(GPR:0 SEL_X:0) MEGA_FETCH_COUNT:31 DST(GPR:2 SEL_X:0 SEL_Y:1 SEL_Z:2 SEL_W:5) USE_CONST_FIELDS:0 FORMAT(DATA:48 NUM:2 COMP:0 MODE:1)
0010 0008000C ENDIAN:0 OFFSET:12
0011 00000000
0002 00000000 CF ADDR:0
0003 85000000 CF INST:20 COND:0 POP_COUNT:0

```

# Consequences to the compiler developer

- CF
  - Turn if/else instructions to execution mask operations
  - Turn while, do..loop and jumps into LOOP\_\*
  - Find instructions to skip with JUMP
- ALU
  - Try to fill all 5 ALU slots
  - Observe all restrictions
  - Vectorize 4 threads into one
- Memory
  - Find the right (=fastest) buffer type
  - Write cache friendly programs
  - Safe memory accessing instructions

## Literature

<http://x.org/docs/AMD/r600isa.pdf>

[http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD\\_Evergreen-Family\\_Instruction\\_Set\\_Architecture.pdf](http://developer.amd.com/sdks/AMDAPPSDK/assets/AMD_Evergreen-Family_Instruction_Set_Architecture.pdf)



**»Wissen schafft Brücken.«**